Network Working Group

Request for Comments #56

Ed Belove (Harvard)

Dave Black (Harvard)

Bob Flegal (Utah)

Lamar G. Farquar (Utah)

June, 1970

Third Level Protocol

Logger Protocol

General Description

In our view of the world each host has a set of four programs to allow a user teletype to communicate with a foreign monitor. The exact implementation of these programs is highly installation-dependent. Thus all explanations are meant to describe functional characteristics rather than design.

The four programs come in two male/female pairs. A user employs a send-logger at his site to communicate with receive-logger at the appropriate foreign site in order to establish a full duplex link between the user's teletype and the foreign machine's monitor. This puts him in the equivalent of a pre-logged in state at the other machine. After the link has been established, the two loggers drop out of the picture, and the user is left talking to a sender in his machine, whose main function is to take input from the user's teletype and send it down the link that was established by the loggers to the receiver in the foreign host, which passes it along to its monitor (making it look

like input from a local teletype). Replies from the foreign monitor are given by it to the receiver, which transmits them back along the link to the sender, which outputs them on the user's teletype. The sender and receiver in each machine must either exist in multiple copies, one for each network user, or there must be a single copy which can handle all of the network users. The loggers, however, need be able to handle only one user at a time, since their task is quickly accomplished, leaving them free to satisfy other requests. However there should be some method of queuing requests that can not be satisfied immediately. A less satisfactory alternative would be to give a busy message to any user who tries to use the logger while it is busy. (This, of course, does not preclude the possibility of an installation having a re-entrant logger, or of having multiple copies of the logger.)

The receive-logger should be user zero in every machine and should always be listening to socket zero. (This same thing can be accomplished by having the NCP intercept all messages to user zero, socket zero, and send them to the receive-logger; but it is simpler and cleaner to have the logger actually be user zero and have the NCP handle its messages the same as everyone else's.)

When the send-logger is called, it pulls a pair of unused sockets (2N and 2N+1) from a pool of free sockets and CONNECT from 2N+1 to User 0, Socket 0 in the desired foreign host. This activates the receive-logger, which accepts the connection if it has an available slot for the foreign teletype. He then immediately closes down this connection to allow links from other sources to be initiated. If, on the other hand, there is no room for the foreign teletype (or if, for

some other reason, the receive-logger does not wish to connect) the attempted link to socket zero is refused. This notifies the send-logger that he cannot log on the foreign host and it then notifies the user of this fact. There is no guarantee, however, that the close was actually sent by the foreign logger. It could have been sent by the NCP if, for example, the pending call queue for the socket was overloaded.

If the link to socket zero has been accepted (thus indicating that the receive-logger can accommodate the request) after closing that link, the receive-logger picks an available pair of sockets (2M and 2M+1) from its pool, and connects from 2M+1 to 2N. (It found the identity of 2N when its listen was answered by the link with 2N+1.) The send-logger has meanwhile listened to socket 2N and now accepts the link, and CONNECTS from 2N+1 to 2M. The receive-logger has been listening to this socket and accepts the attempted link.

At this point, there is a full duplex connection between the two loggers. They then activate the sender and receiver, which handle all other communication between the user and the foreign monitor. (The senders and receivers can be part of the loggers, or can be called by them, etc.)

When the user is finished and escapes back to his monitor, it is up to the sender to close down the links. On the receiving end, it would be highly desirable for the NCP to notify the receiver of this fact, so it could log the user off (if he had failed to do that himself) and could free any resources that he had been using.

A more formal outline of the proposed protocol described in the scenario above follows:

1. Stable state: receive-logger at foreign host listening to User 0, Socket 0.

2. Local user calls send-logger.

3. Send-logger calls CONNECT (port, 2N+1, <foreign host#,0,0>).

4. Send-logger calls LISTEN (port, <local host#, user#, 2N>).

5. Foreign logger's LISTEN is answered, and he is told local user number, host and #2N+1.

6. Foreign logger looks for available sockets (2M and 2M+1). If they exist and it is able to establish connection, it accepts and then immediately closes the link.

7. Foreign logger calls CONNECT (port, 2M+1, <local host#, user#, 2N>).

8. Foreign logger calls LISTEN (port, <local host#, user#, 2M>).

9. Send-logger has listened to 2N and accepts link, then calls CONNECT (port, 2N+1, <foreign host#, user#, 2M>).

10. Receive-logger, which is listening on 2M, accepts link.

11. Loggers activate appropriate handlers.

12. When the user is finished, sender closes down both links.

This basic method of establishing a full duplex connection should be standard throughout the network. The particular way each installation handles the implementation of the sender, receiver, and the two loggers is of no consequence to the network and is highly machine dependent. (Even the fact of needing a sender and receiver is machine dependent in that some members of the network might be able to handle

their functions in other ways.) However, some conventions must be established regarding communication between the sender and receiver, or their equivalents.

Network Standard Code

In order to facilitate use of the network, we propose the convention that all teletype-to-foreign-monitor communication be done using 128 character USASCII. (This is the code used by the IMP's and is in the appendix to the IMP operating manual.) It makes sense to require machines to make only one conversion to a standard code, than to have to make conversions to every code on the net.

In addition, since most of the network machines use ASCII as their internal character code, it will be no trouble for them. Even those machines that use a different code must translate to and from ASCII in order to communicate with local teletypes. Extending this translation to the network should cause very little trouble. We envision this translation as taking place in the sender and receiver, but again that is implementation dependent.

If ASCII is adopted as a standard, we would suggest that all non-ASCII machines create a monitor to the machine's internal code. This would make the complete character set available to those who wished to use it (and were willing to write a simple conversion routine for the local machine.) In this way, those users who wanted to could use any machine on the net from their teletype, without requiring their machines to have records of all the network codes, and yet could use the full power of the foreign machine if they wanted.

Again, this standard applies only for teletype-to-foreign-monitor communication.

Break Characters

A standard way of handling the break character has to be established for the network and be included in the protocol. Problems with the break character arise in several contexts. First, there are two distinct purposes served by the break character. One is as a panic button. This says, "I do not care what is happening, stop and get me out to monitor level now." This command is executed immediately upon receipt, and is most commonly used to get out of a program that one does not want to be in (e.g., one that is in an infinite loop, etc.)

The other purpose that is served is that of an exit from a subsystem, or on a machine with a forking structure as a method to get back to the next higher level fork. This second purpose is not an immediate one in that the user wants the system to finish all that he has told it to do before exiting.

We assume that there does not exist in every system 1) a way of performing each of these functions, or 2) a clear cut distinction between the calling and operation of the two. Furthermore, there are subtle distinctions as to how each system treats the commands.

The panic button function can easily be performed by the proposed control command <INT>. This function must be accomplished by using a control command, since a program can enter a state where it is accepting no input: hence, the program cannot be aborted by sending it a message down the teletype link. There is no reason to worry about the race condition caused by sending this command down the control link since its whole purpose is to force the machine to disregard everything else the user has sent.

In our implementation of this, we would ask the user to specify

to the logger a seldom used character that he wants to be his foreign panic button. Then, it would be a simple task for the sender to map this character into an <INT> command, which the foreign machine must interpret properly. This scheme would work well for most machines, but some may lend themselves to different ways of generating the <INT>.

The other problem that presents itself is what to do if the foreign machine's "exit" character is the same as the local machine's. The problem is that while a user is talking to a foreign machine, he would want to be in a transparent mode, where everything he types is sent directly to the other machine. The way he would get himself out of this mode is to type either his machine's "exit" character or its panic button. Thus, if the foreign machine has the same one, there would be no way to send it. The way out of this is the same as above--merely a mapping of another seldom used character into the foreign machine's "exit" character. This type of mapping can be carried as far as each installation deems necessary. Giving the user complete control over translation is helpful in that it allows him to user characters that his teletype cannot generate.

Command Message Formats

Each site should establish its now conventions about when to send a monitor command string, and in what size chunks. When performing a routine operation, one might want to send several command lines as a single message. If working with the monitor as usual, a reasonable break point might be at every carriage return. When using a highly interactive language such as QED, one might decide character-

to the logger a seldom used character that he wants to be his foreign panic button. Then, it would be a simple task for the ATTNhandler to map this character into an <INT> command, which the foreign machine would interpret properly. This scheme would work well for most machines, but some may lend themselves to different ways of generating the <INT>.

The other problem that presents itself is what to do if the foreign machine's "exit" character is the same as the local machine's. The problem is that while a user is talking to a foreign machine, he would want to be in a transparent mode, where everything he types is sent directly to the other machine. The way he would get himself out of this mode is to type either his machine's "exit" character or its panic button. Then, if the foreign machine has the same one, there would be no way to send it. The way out of this is the same as above--merely a mapping of another seldom used character into the foreign machine's "exit" character. This type of mapping can be carried on for as each installation deems necessary. Where the more complete control over translation is helpful is that it allows him to send characters that his teletype cannot generate.

Command Message Formats

Each site should establish its own conventions about when to send a complete command string, and in what size chunks. When performing a sequential edit, one might want to send several command lines as a single message. If working with the monitor as usual, a reasonable break point might be at every carriage return. When using a highly interactive language such as QED, one might wish to character-

by-character transmission was a necessity. We feel that each user should have the choice between these three methods (and possible more). Furthermore, the user should be able to change between each mode at will. The differences in syntax of the send-message commands mentioned above should be noted. For the first, a special send-message command character must be defined, and it should not be sent along with the message. For the second, the carriage return acts dually as the send-message command and as a command delimiter. Therefore it must be sent with the message. Finally, the case of character-by-character transmission with its implicit send command should pose no significant problems.

The preceding discussion is meant to imply also that the receiver must be able to buffer up each of the above types of transmission into a form acceptable to its own monitor interface.

In addition, all echoing should be done in the local host, with the foreign machine suppressing its echoes (if it can).

We would like to thank Carl Ellison (of Utah) for his valuable suggestions and criticisms of this work, and Jim Curry (of Utah) for his encouragement and support of the effort.